jamk.fi

Security Automation for Windows Hosts: Hardening of Windows 10 Password Policy

Ronald Clark

Master's thesis
April 2020
School of Technology
Information and Communication Technology
Master of Engineering, Cyber Security

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Description

Author(s)	Type of publication	Date
Clark, Ronald	Master's thesis	May 2020
		Language of publication: English
	Number of pages: 57	Permission for web publication: x

Title of publication

Security Automation for Windows Hosts: Hardening of Windows 10 Password Policy

Degree program

Master of Engineering, Cyber Security

Supervisor(s)

Rantala, Ari

Assigned by

Abstract

Security automation using configuration management tools is not a new thing. These tools can be used in many ways. One of the most common uses of configuration management tools is to keep system settings consistent across like systems. Many times, the focus is not security, but other general host or node configurations. The focus of this project was to see if Ansible, a configuration management tool, is a viable tool for security automation on Windows Hosts.

Ansible was built and designed to support configuration management for Unix or Linux based systems. The idea of Ansible as a configuration management tool used to perform security automation on a Windows 10 hosts is not so common. Information was gathered on Ansible, Windows, Security Frameworks as well as other tools, and used to build a security baseline based on NIST SP 800 53r4. The security baseline was limited to the password policy of the Windows host.

The security baseline of controls and values for those controls were developed, then the manual method of implementing the security controls was investigated, and finally an Ansible script was developed to automate this manual implementation. The script contained a compliance and configuration section. The compliance section checked the running configuration on the Windows host to verify if it was in-line with the security baseline and alerted if some control was non-compliant.

The configuration portion corrected any non-compliant controls so that the controls would meet the compliant security baseline. All the controls operated in the way they were designed. The security automation was a success.

Keywords/tags (subjects)

Security automation, Ansible, Windows, NIST, configuration management, hardening

Miscellaneous (Confidential information)

Contents

Figures

Tables

Acronyms

NIST	National Institute of Standards and Technology
CM	Configuration Management
IT	Information Technology
INI	Initialization
YAML	Yet Another Markup Language
VM	Virtual Machine
OS	Operating System
ISO	International Standards Organization
COBIT	Control Objectives for Information and Related Technology
CLI	Command Line Interface
SOAP	Simple Object Access Protocol
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
WinRM	Windows Remote Management

1 Introduction

1.1 Background

This document has many purposes, one of which is to demonstrate how a security baseline can be developed based on the latest and most advanced security frameworks. Another purpose is to demonstrate how this security baseline can be automated to quickly assess the security posture of a network element and correct any inconsistencies within those network elements. Lastly, this document shows how these corrections can be done with simple configuration management tools that do not require complex algorithms or an advanced setup.

The benefits of this document include knowledge about automating some aspects of security management, knowledge about using configuration management tools for Windows machines and mapping security controls to security frameworks. The lay user can get an understanding of the process and the more advanced user can adapt this methodology and implement usage into their infrastructure to help make some aspects of security more simplified.

This document is suitable for any audience who wants to learn about using configuration management and automation to help enforce security. More specifically, Network professionals, Security professionals, and System Administrators can benefit from the information in this document.

1.2 Project Goal

The goal of this project was to expand and share knowledge of automation, and show that Ansible is a viable tool for security configuration management on Windows hosts. There was some basic understanding and practical experience with automation using configuration management tools in place, but this only applied to Linux environments. Learning how this works with Windows machines extended the current knowledge and made the application more versatile. Hopefully this project

can help a reader to gain some insight on these topics and increase their comfort level when using these tools.

A success for this project could be defined by the clarity of which the information was presented, the ease of its understanding, and the confirmations that the methods worked. An individual reading this document should be able to say that the information was clear and easy to understand. Many times, documents contain advanced technical jargon, which makes understanding difficult for beginners. The point of explaining something to others is not to raise more questions, but to answer the questions with clarity and make understanding of the subject as simple as possible. Also, this document should prove that the shown methods succeeded in their intended goals. This document shows how configuration management tools can be used to automate security hardening. It also shows that the hardening works when tested. If these above items are achieved, then the goals of this document will be reached.

1.3 Risks

There was one big risk to this project. During the time when this project was being reported and tested, the world was in the midst of a pandemic. The Coronavirus pandemic put a stop to many social and economic activities. This included social distancing in order to prevent the spread of disease which made gathering sources of information challenging because there was no access to public or private library facilities. All schools and universities were closed during this time.

2 Research Methods

2.1 Research Summary

This project required research to get an understanding of how to execute the technical parts. There are two main methods of performing research and those are qualitative and quantitative. Both methods are effective but the approach to each method and ways of gathering data is different. The nature of this project did not require any numerical data collection or comparisons. If this were the case the

research would have taken a quantitative approach. This project required a descriptive type of data. Examples of these are interviews, protocols, written records, and videos (Cropley 2019). The method of research that is in line with these types of information is qualitative. The research for this project took a qualitative approach.

The other research method that was used in this project was benchmarking. Benchmarking is an effective way to use comparison. It can show how the traits of one item versus the traits of another can be more or less beneficial. This can help to drive decision making toward the more useful item or principles.

2.2 Qualitative Research

Qualitative Research is scientific research that seeks to answer a question, uses a predefined procedure for question answering, collects evidence, helps produce new findings, and produces findings that apply beyond the boundaries of the study (Mack & Woodsong 2005). Additionally, it seeks to understand a given research problem or topic from the perspectives of the parties involved (Mack & Woodsong 2005).

There are three commonly used qualitative methods. These methods include in-depth interviews, participant observation and focus groups (Mack & Woodsong 2005). The methods are explained in Table 1 below.

Table 1. Three Common Qualitative Methods (Mack & Woodsong 2005)

Method	Description
Participant Observation	Good when collecting data on behaviors occurring in their normal contexts
In-depth Interviews	Appropriate when collecting data of individuals' personal histories, perspectives, and experiences, with regard to the topic being explored.
Focus Groups	Effective in eliciting data on the cultural norms of a group and in generating broad overviews of issues of concern to

	the cultural groups or subgroups represented.

These are the most common qualitative methods but there are also other less common methods used within Qualitative Research.

Qualitative data takes on the forms of field notes, recordings, videos and transcripts (Mack & Woodsong 2005). These are effective ways to maintain the integrity of the data that has been received. An accurate analysis can be completed on this data because the data is not misrepresented in the form of reciting from memory or other less reliable forms of data capturing. The qualitative methods used for this project helped to define the test environment, affirm the choice of which configuration management tool to use, and how to construct the security baseline.

2.3 Interview

Interviews are a great way to get direct insight from a subject. To conduct an interview in a qualitative way, the questions must be open-ended, so they allow for elaboration from the interviewee. The interview can take on many forms. The questions can be a set list that has been predefined or the questions can target a specific group of areas (Showkat & Parvenn 2017). These types of interviews are structured and semi-structured. There are also unstructured interviews that take the form of a conversation of a particular topic (Showkat & Parvenn 2017). The interview that was conducted for this project was an unstructured conversational interview with a colleague Security DevOps Engineer Sharma on 18 March 2020 who has knowledge in the area. The interview provided some insight on the technical environment needed to execute this project, how to choose and map security controls and how to verify the hardening. This interview proved to be a helpful source in the journey to answer the question this project intended to answer. Her advice to use VirtualBox to host the test environment matched the interviewer's personal opinion to use this. Also, her explanations of why Ansible would work well left no doubts about using Ansible for this project.

2.4 Benchmarking

What is benchmarking and how does it apply to this project?

Benchmarking is the process of locating good practice and learning from others (Fice & Waller 2012). Table 2 below, shows the different types of benchmarking that can be found in existing literature on this subject.

Table 2. Benchmarking Types (Fice & Waller 2012)

Types of Benchmarking
Implicit (by-product of information gathering) or explicit (deliberate and systematic)
Conducted as an independent (without partners) or a collaborative (partnership) exercise
Confined to a single organization (internal exercise), or involving other similar or dissimilar organizations (external exercise)
Focused on an entire process (vertical benchmarking) or part of a process as it manifests itself across different functional units (horizontal benchmarking)
Focused on inputs, a process or outputs (or a combination of these)
Based on quantitative and/or qualitative information

Benchmarking was used in this project to determine the best practice values of the selected security controls that were used to harden the target host. The cyber security framework from the National Institute of Standards and Technology (NIST) was used as a benchmarking target.

3 Concepts and Associated Tools

3.1 Tools and Concepts Summary

This section describes the concepts such as configuration management, virtualization, and security frameworks. It also describes some of the tools that were used to create the environment for the project. These tools include Ansible, Virtual Box, Windows 10 virtual machine, Kali Linux virtual machine, and the NIST cyber

security framework. An overview of these concepts and tools will be provided in the following sections to help give a better understanding of their purpose in general and their purpose in respect to this project. Table 3 below shows these tools and concepts briefly.

Table 3. Description of Concepts and Tools

Concept/Tool	Purpose
Configuration Management	Managing any configurations on any host machine. Example: Setting the size of audit logs and ensuring the setting does not change.
Virtualization	Concept of working with virtual machines instead of physical machines
Security Frameworks	Used to establish security baseline and controls
Virtual Box	Virtualization tool to create environment
Virtual Machine	Virtualized form of node and operating system
NIST	Mapping of security controls

3.2 Configuration Management

Configuration management (CM) can be defined in many ways depending on the context in which the term is used. In this context, it is defined as a process of systematically handling system changes so that the system maintains its integrity over time (Heidi 2019). The term did not originate from information technology (IT) but has become linked to the management of configurations on servers.

The role of automation is huge within configuration management. Automation is used to make and keep server(s) at the desired state (Heidi 2019). In order to implement automation, scripts are developed in some markup or programming

language and are used to perform this CM automatically. Automation and CM work hand in hand. Automation is the driving force or heart of CM.

Server Orchestration is another common term used when discussing CM and automation. Server Orchestration tools are capable of managing from a few to many servers with one control machine (Heidi 2019). Popular tools used for CM include Salt, Ansible, Chef, and Puppet. They all have their various ways of achieving the server's optimal state but their purpose is basically the same.

Below in Table 4, some benefits of CM are listed.

Table 4. Benefits of Configuration Management (Heidi 2019)

Benefit	Description
Quick Server Provisioning	New servers can quickly be deployed and configured to the desired state in minutes instead of hours.
Quick Recovery from Critical Events	Because of quick provisioning, troubleshooting damaged servers can be troubleshot without rush because replacement servers can be quickly deployed during the troubleshooting process.
Snowflake Servers no longer exist	Manual hotfixes/configurations are hard to maintain and create unique server configurations that are hard to track. Automation relieves this issue and keeps track of all changes to servers.
Version Control for Servers	Tools such as Git can be used to track the versions of the server provisioning scripts, therefore the configuration versions of the servers can also be tracked.

Replicated Environments	When production, development and testing environments need to be identical, CM and automation can ensure that there are no discrepancies between these environments.

There are a few things to take into consideration when deciding which tool to use for CM. Table 5 below lists some of these things and provides a brief explanation.

Table 5. Things to Consider When Choosing a CM Tool (Heidi 2019)

Consideration	Questions to Answer
Infrastructure Complexity	Which tool is right for the complexity of my infrastructure? Will this tool increase the complexity?
Learning Curve	How difficult is installation and use of this tool?
Cost	How expensive would it be to use this tool?
Advanced Tooling	Is there a need for extensions to the tool? Are more advanced features needed in addition?
Community Support	How well is the tool documented and maintained? Is the tool sustainable for the future?

3.3 Ansible

Ansible is one of the CM tools mentioned in the previous section. Ansible is an IT automation/CM tool that can configure systems, deploy software, and orchestrate other more advanced IT tasks (Red Hat 2019). The main goal with Ansible is simplicity and ease-of-use. This flattens the learning curve of the tool and usage can begin within minutes. It also has a strong focus on security and reliability, featuring a minimum of moving parts, usage of OpenSSH for transport (with other transports

and pull modes as alternatives), and a language that is designed around auditability by humans–even those not familiar with the program (Red Hat 2019).

Some common terminology in Ansible and its meaning can be seen below in table 6

Table 6. Ansible Terminology (Heidi 2019)

Term	Description
Control Node	Machine where Ansible is installed and is used to manage all other nodes
Inventory	INI file of all the Ansible targets to be managed
Playbook	YAML file with a series of procedures to be automated
Task	One block that defines a procedure to be executed
Module	All Ansible tasks are built with modules that abstract system actions such as copying files or restarting a service.
Role	A set of related playbooks, templates and other files, organized in a pre-defined way to facilitate reuse and share
Play	The complete execution of a playbook from start to finish
Facts	System information about the Ansible target gathered by the Control Node
Handlers	Used to trigger service status changes, e.g. restarting or reloading a service

The above terminology is essential in understanding Ansible and its usage.

The format of a typical ansible playbook is displayed in Figure 1.

```yaml
---
- hosts: localhost
  become: true
  gather_facts: false

  vars:
    contexts:
      value: "local context2"

    services:
      value: "Telnet ftp ssh http"

    user_vars:
      - contexts
      - services

  tasks:

    - debug:
        msg: "{{ item[0] }}  {{ item[1] }}"
      with_nested:
        - "{{ contexts.value.split(' ') }}"
        - "{{ services.value.split(' ') }}"
```

Figure 1. Example Ansible Playbook

In the above example there are various parts to this example file. Table 7 below explains each part of the Ansible Playbook shown in Figure 1.

Table 7. Ansible Playbook Breakdown

Playbook Item	Description
hosts	Can have the setting All or any specific host targeted by the playbook for management
become	Boolean option, can only be true or false, tells whether Ansible is allowed to escalate to a privileged user if needed to complete some task(s)
gather_facts	Boolean option, can only be true or false, informs Ansible if it should gather information about the target system
vars	List of global variables used in playbook
tasks	Beginning of the section where all of the Ansible tasks are defined for the playbook
debug	An Ansible module used to display messages or other information within the playbook

As mentioned earlier in this section, Ansible has many modules that can be used to execute tasks in playbooks. The ansible documentation by Red Hat is a good place to gain more knowledge about Ansible.

3.4 Virtualization

Virtualization allows the creation of useful IT services that are traditionally bound to hardware. A physical machine's full capacity could be used by distributing its capabilities between users or environments (Red Hat 2020).

An example of a traditional setup where one server has 1 task so 3 servers would be needed to have one function as a web server, another as mail server and the last hosts some applications. Figure 2 below shows this setup.

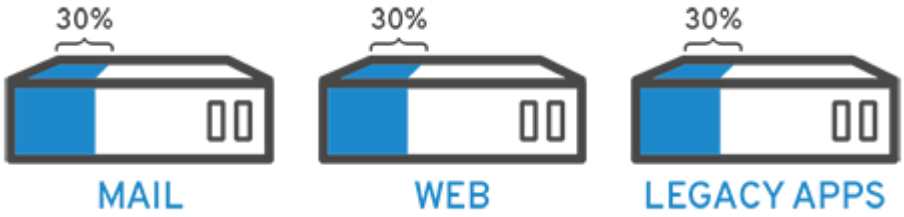

Figure 2. Servers Without Virtualization (Red Hat 2020)

If 2 of the servers have the same hardware configurations, let's say the mail server and the server running the applications, then with virtualization, there would be no need for 3 servers. The applications and mail services could run on the same server as 2 separate unique entities as pictured below in Figure 3.

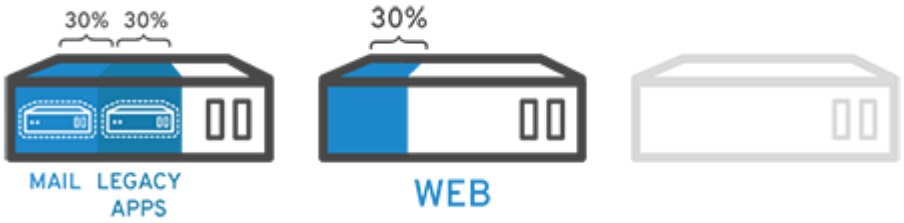

Figure 3 Server 1 With Virtualization (Red Hat 2020)

This would create a more cost effective and efficient environment. Instead of 3 servers operating at 30% each, now there are 2 servers with one operating at 60% and one at 30%.

Virtualization works by using software called hypervisors that separate the physical resources of a server from the virtual environments (Red Hat 2020). Hypervisors can be implemented in a few different ways. One way is to run it on top of an operating system (OS) like it is done with laptops (Red Hat 2020). Another way is to run it directly on the hardware like it is done with servers (Red Hat 2020). The second of the two options mentioned, is how most enterprises implement hypervisors. The hypervisor will divide physical resources so that virtual resources can use them (Red Hat 2020).

“Resources are partitioned as needed from the physical environment to the many virtual environments. Users interact with and run computations within the virtual environment (typically called a guest machine or virtual machine). The virtual machine functions as a single data file. And like any digital file, it can be moved from one computer to another, opened in either one, and be expected to work the same. (Red Hat 2020)” Figure 4 shows the hypervisor’s place between the virtualizations and hardware.

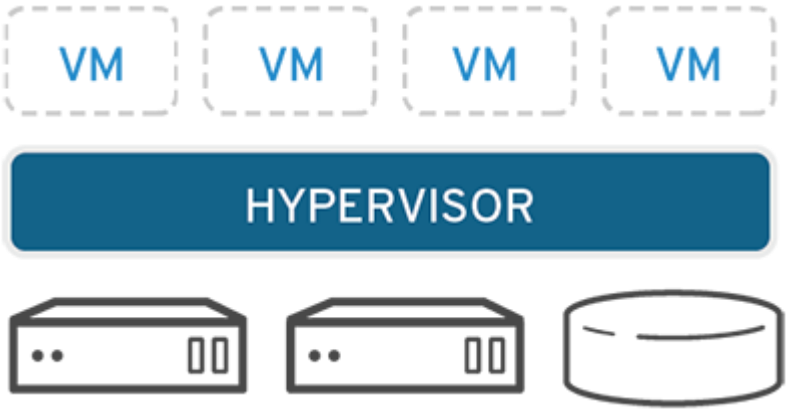

Figure 4. Location of Hypervisor (Red Hat 2020)

What are the types of virtualization that exist? Table 8 shows the types of virtualization and provides a brief description of them.

Table 8. Types of Virtualization

Type of Virtualization	Description
Data Virtualization	Treats data as a dynamic supply that can be fed to chosen resources
Desktop Virtualization	Allows the viewing, configuration and monitoring of physical desktops all to 1 desktop by launching virtual forms of the physical desktops
Server Virtualization	Virtualizing a server lets it to perform more of those specific functions and involves partitioning it so that the components can be used to serve multiple functions (Red Hat 2020).
Operating System Virtualization	Allows running of multiple operating systems side by side on one physical machine.
Network Function Virtualization	Separates a network's key functions (e.g. directory services, file sharing, and IP configuration) so they can be distributed among environments. Once software functions are independent of the physical machines they once lived on, specific functions can be packaged together into a new network and assigned to an environment. Virtualizing networks reduces the number of physical components—e.g. switches, routers, servers, cables, and hubs—that are needed to create multiple, independent networks, and it is particularly popular in the telecommunications industry (Red Hat 2020).

3.4.1 Virtual Box

VirtualBox is a powerful virtualization product for home and enterprise use. Virtual machines of many types can be deployed in VirtualBox. VirtualBox can also run on many host environments. It runs on Windows OS/Servers, Linux, Solaris, OpenBSD, etc. (VirtualBox.org 2020). VirtualBox is the only professional solution that is Open Source Software and is available for free to all users (ibid.). VirtualBox is continuously maintained and developed adding features, supported guest OS and platforms that it can run on (ibid.). The environment for this project is housed completely within

VirtualBox using VMs. VirtualBox easily supports this environment and makes executing this project simple.

3.4.2 Virtual Machine (VM)

A virtual machine is an image file that runs on a host machine. It behaves like a host machine, sort like having a computer within a computer. It runs in its own window giving the same user experience as when using the host. Virtual machines are isolated from the host machine. The software they run cannot reach the components of the host machine which makes it ideal for many things, e.g. testing new software, testing updates, backups, and analyzing malware. These are all benefits of the sandboxed VM that is being run on the host.

A host is not limited to one VM. A host can run many different VMs that are running very different software or applications. Each virtual machine provides its own virtual hardware, including CPUs, memory, hard drives, network interfaces, and other devices (Microsoft 2020). The virtual hardware is mapped to the physical hardware of the host, which saves costs for the user.

3.5 Security Framework

An information/cyber security framework is a set of guidelines or best practices that an organization can adapt to secure its infrastructure and other assets. There are many security frameworks; however, a few of them are more prominent than others. Some of the more recognized ones include International Standards Organization (ISO) 27K, Control Objectives for Information and Related Technology (COBIT), and US National Institute of Standards and Technology (NIST). In addition to those, there are some industry specific frameworks that target sectors such as health care and payment systems. The idea of having these frameworks is to provide a guideline for protecting the confidentiality, integrity and availability of all data and other assets.

The security frameworks list various security controls and define how to meet compliance with the listed controls. This project focused on the NIST security framework. The controls that were adapted were taken from NIST and the compliance was based on the values that are defined in NIST.

3.6 NIST

As mentioned above, NIST stands for National Institute of Standards and Technology. It is the most prominent security framework in the United States. Most government entities and many other private companies in the USA use NIST as their security framework. Some history about NIST Cyber Security Framework is that it was a government initiative in February 2013 (NIST 2019). The US government needed a set of security guidelines to protect the critical infrastructure from cyber attacks. An executive order from the President directed NIST to work on a voluntary framework based on existing standards and practices (NIST 2019).

Through the help of government and the private sector, the formation of the Cyber Security Framework was put into to action. This framework is a voluntary guidance, based on existing standards, guidelines, and practices so organizations can better manage and reduce cybersecurity risk (NIST 2019). It is comprised mainly of 3 components. Figure 5 shows the main components of the framework.

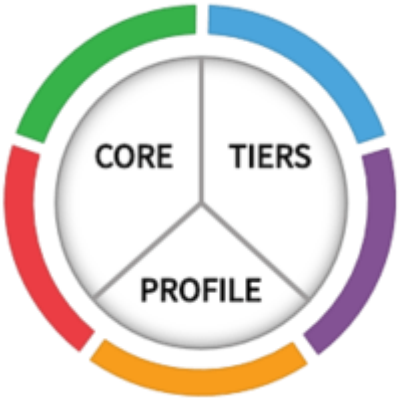

Figure 5 Main Components of NIST Cyber Security Framework (NIST 2019)

The Core of the framework defines a set of the desired cybersecurity activities and outcomes using understandable language so that the guidelines can easily be followed (NIST 2019). The idea is to complement the existing security and risk management policies of companies. The Framework Implementation Tiers help by providing context on how an organization views cybersecurity risk management (NIST 2019). “The Tiers guide organizations to consider the appropriate level of rigor

for their cybersecurity program and are often used as a communication tool to discuss risk appetite, mission priority, and budget" (NIST 2019). Finally, the framework Profiles are the organization's alignment of their requirements and objectives, risk appetite, and resources against the outcome of the Framework Core (NIST 2019). These Profiles are used to improve cybersecurity at organizations by identifying and prioritizing areas of needed improvement.

4 Automating Implementation of Windows Security Baseline

4.1 Timeline

This project began in February 2020 and was completed in May of 2020. The timeline includes outlining and submitting the project, the beginning of the project documentation, the execution of the project and the final project documentation. Table 9 gives a more descriptive picture of the timeline for this project.

Table 9. Timeline for Project

Start	Items	End
February 2020	Topic outline and request for approval	February 2020
March 2020	Gathering sources and initial theory documentation	March 2020
April 2020	Technical implementation and Documentation	April 2020
April 2020	Final Documentation and submitting	May 2020

There was a strict deadline of completing this project by 4 May 2020. The only challenges to the timeline were full time work and school courses running in parallel. It was a big challenge to balance everything, but the timeline was followed, and the project did not go past the deadline.

4.2 Environment Setup

The technical environment was setup in VirtualBox. The first thing that was needed after the installation of VirtualBox was to setup the VMs that would be used in the project. One of the VMs used in the project was an OVA file containing an instance of Kali Linux. The process required that the OVA file was downloaded to the host machine and then imported to VirtualBox as an appliance. Figure 6 shows the Kali Linux OVA file being imported to VirtualBox.

Figure 6. Kali Appliance Imported to VirtualBox

Once the appliance is imported, the Kali VM is ready for operation. The tool used to automate the security baseline was Ansible. The reason the Kali VM was chosen was because Ansible cannot be installed on a Windows host. It needs a Unix/Linux OS to run on and act as the Ansible Machine.

Next came the installation of the Windows VM which served as the target for security hardening. The Windows VM derived from an OVA file as well. The same steps were repeated to install the Windows VM. As mentioned above, Ansible is the tool that was needed for the automation process. This required Ansible to be installed on the Kali VM. Ansible was installed by opening the Command Line Interface (CLI) and using the command: apt-get install ansible. This began the installation and Figure 7 shows the Ansible version that was installed.

Figure 7. Ansible Installed on Kali VM

Ansible usually connects to targets via SSH, but this is not possible for Windows hosts. Ansible uses WinRM to communicate with Windows hosts. WinRM is a management protocol that Windows uses to connect to remote servers (Red Hat 2019). This protocol is SOAP based and communicates over HTTP/HTTPS. It is included in all the most recent Windows OS (Red Hat 2019). More specifically, Ansible uses the pywinrm package to communicate with Windows hosts over WinRM (Red Hat 2019). The pywinrm package does not come with the default installation of Ansible, so it requires the installation of this package. The pywinrm package was successfully installed so the Ansible Machine was ready for communication with the Windows target. After one attempt to connect to the Windows target, there were some extra settings that needed to be adjusted on the Windows target to allow communication. WinRM was installed on Windows but it was not turned on. A script was run on the Windows target to turn on WinRM and the script was received from (Henderson 2018). There was still one issue when trying to execute the PowerShell script. Figure 8 below shows the error message that was presented when the script was executed.

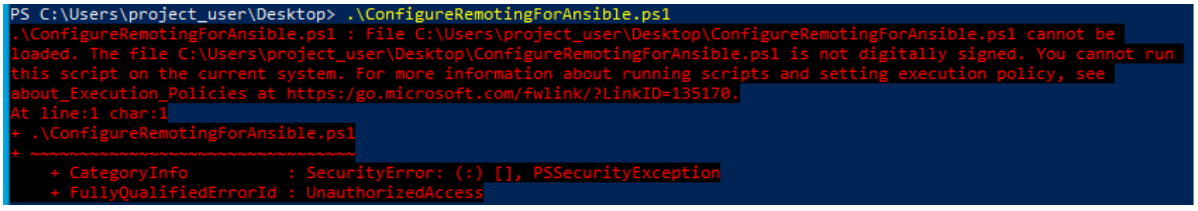

Figure 8. Error Presented from WinRM PowerShell Script

This error was pointing to PowerShell's execution policy. Table below gives a description of the PowerShell Execution Policy options.

Table 10. PowerShell Execution Policy Options (Wheeler 2020)

Policy Type	Description	Policy Scope	Description
AllSigned	Scripts can run. Requires that all scripts and configuration files be signed by a trusted publisher, including scripts that are written on the local computer. Prompts the user before running scripts from publishers that have not yet classified as trusted or untrusted. Risks running signed, but malicious, scripts.	MachinePolicy	Set by a Group Policy for all users of the computer.
ByPass	Nothing is blocked and there are no warnings or prompts. This execution policy is designed for configurations in which a PowerShell script is built into a larger application or for configurations in which PowerShell is the foundation for a program that has its own security model.	UserPolicy	Set by a Group Policy for the current user of the computer.
Default	Sets the default execution policy. Restricted for Windows clients. RemoteSigned for Windows servers.	Process	The Process scope only affects the current PowerShell session. The execution policy is saved in the environment variable

			$env:PSExecutionPolicyPreference, rather than the registry. When the PowerShell session is closed, the variable and value are deleted.
RemoteSigned	The default execution policy for Windows server computers. Scripts can run. Requires a digital signature from a trusted publisher on scripts and configuration files that are downloaded from the internet which includes email and instant messaging programs. Does not require digital signatures on scripts that are written on the local computer and not downloaded from the internet. Runs scripts that are downloaded from the internet and not signed, if the scripts are unblocked, such as by using the Unblock-File cmdlet. Risks running unsigned scripts from sources other than the internet and signed scripts that could be malicious.	CurrentUser	The execution policy affects only the current user. It's stored in the HKEY_CURRENT_USER registry subkey.
Restricted	The default execution policy for Windows client computers.	LocalMachine	The execution policy affects all

	Permits individual commands, but does not allow scripts. Prevents running of all script files, including formatting and configuration files (.ps1xml), module script files (.psm1), and PowerShell profiles (.ps1).		users on the current computer. It is stored in the **HKEY_LOCAL_MACHINE** registry subkey.
Undefined	There is no execution policy set in the current scope. If the execution policy in all scopes is Undefined, the effective execution policy is Restricted, which is the default execution policy.		
Unrestricted	The default execution policy for non-Windows computers and cannot be changed. Unsigned scripts can run. There is a risk of running malicious scripts. Warns the user before running scripts and configuration files that are not from the local intranet zone.		

The LocalMachine was assigned to the AllSigned policy as shown in Figure 9.

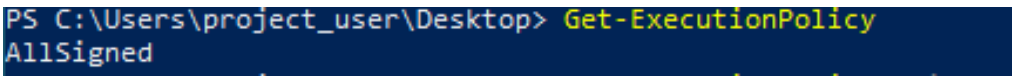

Figure 9. Execution Policy for Local Machine

Table 10 shows that the AllSigned execution policy requires that all scripts be signed by a trusted publisher. The user created for this project; project_user, was required

on the list of trusted publishers. When this was completed, the script was executed successfully. The next item needed for the communication was an inventory file. Ansible requires a file listing required parameters to make connections to targets. This file is called the Ansible Inventory File. Figure 10 is a snapshot of the inventory file.

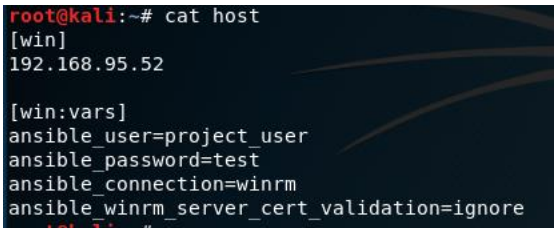

Figure 10. Ansible Inventory File

After the inventory file was created, the connection to the Windows host was attempted for a second time and it was successful. Figure 11 displays this successful connection.

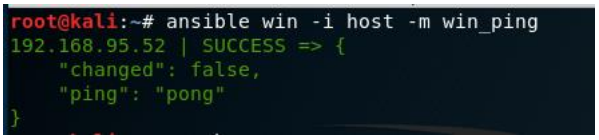

Figure 11. Ansible Connected to Windows Target

The Windows machine did not require any additional software to be installed for communication with Ansible; however, for the purposes of the project, an administrative user was created on the Windows target to be used as the Ansible user when Ansible needed to authenticate to Windows. Also, there was only the need to have communications between the two VMs; therefore, the network configuration in VirtualBox was changed to host only network. This prevented unwanted traffic from unsecure networks from reaching the VMs.

4.3 Mapping Security Controls to NIST

Mapping security controls to the NIST cybersecurity framework first required establishing a set of security controls that should be implemented. This set of

controls and values for the controls are the security baseline. The security controls that were implemented are related to access controls and password policy. The following subsections describe these controls and explains how they are mapped to the corresponding controls in NIST. The mapping for these controls is based on NIST SP 800-53r4 security controls and the baseline values are taken from NIST SP 800-63b.

4.3.1 Control 01: Limit Failed Login Attempts

NIST Mapping: Control: AC-7 Unsuccessful Logon Attempts

Description:

- This control enforces a limit of 10 consecutive invalid logon attempts by a user during a 60-minute time period; automatically locks the user's account for 60 minutes and locks the account until released by an Administrator(NIST 2015).

- "This control applies regardless of whether the logon occurs via a local or network connection. Due to the potential for denial of service, automatic lockouts initiated by information systems are usually temporary and automatically release after a predetermined time period established by organizations. If a delay algorithm is selected, organizations may choose to employ different algorithms for different information system components based on the capabilities of those components. Responses to unsuccessful logon attempts may be implemented at both the operating system and the application levels." (NIST 2015)

4.3.2 Control 02: Enforce Password Complexity

NIST Mapping: Control: IA-5(1(a, b, c)) Authenticator Management

Description:

- This control enforces the minimum password complexity of, minimum length 14, minimum numbers 1, minimum lowercase letters 1, minimum uppercase letters 1, and minimum special characters 1 (NIST 2015).

- Enforces at least 3 changed characters when new passwords are created (NIST 2015).

- Stores and transmits passwords that encrypted only (NIST 2015).

- "This control enhancement applies to single-factor authentication of individuals using passwords as individual or group authenticators, and in a

similar manner, when passwords are part of multifactor authenticators. This control enhancement does not apply when passwords are used to unlock hardware authenticators (e.g., Personal Identity Verification cards). The implementation of such password mechanisms may not meet all of the requirements in the enhancement. Cryptographically protected passwords include, for example, encrypted versions of passwords and one-way cryptographic hashes of passwords. The number of changed characters refers to the number of changes required with respect to the total number of positions in the current password. Password lifetime restrictions do not apply to temporary passwords. To mitigate certain brute force attacks against passwords, organizations may also consider salting passwords." (NIST 2015)

4.3.3 Control 03: Prevent Excessive Password Ageing

NIST Mapping: Control: IA-5 (1(d, e)) Authenticator Management

Description:

- This control enforces a password minimum lifetime of 7 days and a maximum lifetime of 90 days (NIST 2015).
- Prevents password reuse for 10 generations (NIST 2015).
- "This control enhancement applies to single-factor authentication of individuals using passwords as individual or group authenticators, and in a similar manner, when passwords are part of multifactor authenticators. This control enhancement does not apply when passwords are used to unlock hardware authenticators (e.g., Personal Identity Verification cards). The implementation of such password mechanisms may not meet all of the requirements in the enhancement. Cryptographically protected passwords include, for example, encrypted versions of passwords and one-way cryptographic hashes of passwords. The number of changed characters refers to the number of changes required with respect to the total number of positions in the current password. Password lifetime restrictions do not apply to temporary passwords. To mitigate certain brute force attacks against passwords, organizations may also consider salting passwords." (NIST 2015)

4.4 Compliance of Security Baseline

The compliance of the security baseline is one portion of the security automation. In order to verify compliance of a security control, the node was checked for the existing security setting. If that security setting met the security baseline established value(s), then the node was compliant with that control. In the following sections, a

technical description is given of how the compliance was checked for each security control.

4.4.1 Limit Failed Login Attempts

Parameter: Lockout Duration: 60

Description: The amount of time in minutes a local user account is locked before it is automatically unlocked.

Manual Compliance Check: Open PowerShell as an Administrator and issue the command: *net accounts | findstr -i 'duration'.* Figure 12 shows the output from PowerShell.

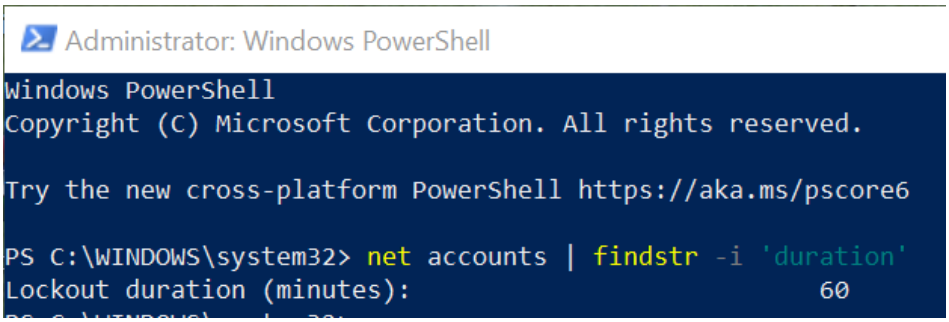

Figure 12. Checking Lockout Duration

Figure 12 shows the current setting for Lockout Duration on the Windows host. The value is always set in minutes and if the value is not set to the defined parameter value taken from NIST, the value is non-compliant and needs to be corrected to the corresponding compliant value.

Automation: The automation of this check was slightly more specific. The idea was to isolate the parameter value only and store it into a variable so this could be used to check against the compliant value. Figure 13 shows the Ansible task used to achieve this.

```
- name: "Retrieve Current Lockout duration Time"
  win_shell: net accounts | findstr -i 'duration' | %{$_.split(':')[1]} | %{$_ -replace ' ', ''}
  register: current_lockout_time
  changed_when: False
```

Figure 13. Ansible Task to Find Running Lockout Duration Value

The Ansible task shown in Figure 13 stores the running configuration of the Lockout Duration into the variable *current_lockout_time*. This variable was used to compare against the security baseline parameter value stored in the variable *Lockout_duration* to determine if compliance has been met or not. Figure 14 shows the Ansible task that makes this comparison.

```
- name: "Fail if Lockout Duration is not {{ Lockout_duration.value }}"
  fail:
    msg: >
     "Security Baseline is not correct. The lockout duration value found: {{ current_lockout_time.stdout |trim }} is not \
      compliant to the expected: {{ Lockout_duration.value }}"
  when: current_lockout_time.stdout |trim |int != Lockout_duration.value
```

Figure 14. Ansible Task Fails if Compliance Not Met

The if the running configuration on the Windows host was not set to 60 minutes, then the Ansible task in Figure 14 failed stating that the running configuration is not compliant.

Parameter: Lockout Threshold: 10

Description: The number of failed login attempts allowed before the user account is locked

Manual Compliance Check: Open PowerShell as an Administrator and execute the following command: *net accounts | findstr -i 'threshold'*. The output of the command is displayed in Figure 15.

```
PS C:\WINDOWS\system32> net accounts | findstr -i 'threshold'
Lockout threshold:                                     5
PS C:\WINDOWS\system32>
```

Figure 15. Checking Lockout Threshold

Figure 15 shows the current running configuration for Lockout Threshold on the Windows host. The value is compliant if it matches the Lock Threshold parameter value of 10 set in the security baseline.

Automation: To automate this check, the current running configuration for Lockout Threshold was extracted from the Windows host and stored to the variable *current_threshold.* Figure 16 shows the Ansible task used to do this.

```
- name: "Retrieve Current Lockout Threshold"
  win_shell: net accounts | findstr -i 'threshold' | %{$_.split(':')[1]} | %{$_ -replace ' ', ''}
  register: current_threshold
  changed_when: False
```

Figure 16. Current Lockout Threshold Captured

The Ansible task shown in Figure 16 extracted only the value so it could be directly compared to the security baseline value. This extracted value was stored in the variable *current_threshold.* Figure 17 shows the Ansible task used to make the direct comparison between the compliant security baseline value stored in the variable *Lockout_threshold* and *current_threshold.*

```
- name: "Fail if Lockout Threshold is not {{ Lockout_threshold.value }}"
  fail:
    msg: >
      "Security Baseline is not correct. The lockout threshold value found: {{ current_threshold.stdout |trim }} \
      is not compliant to the expected: {{ Lockout_threshold.value }}"
  when: current_threshold.stdout |trim |int != Lockout_threshold.value
```

Figure 17. Ansible Task Used to Check Compliance

The task in Figure 17 failed if the *current_threshold* was not set to the security baseline value.

Parameter: Lockout Observation Window: 60

Description: This parameter defines the amount of time in minutes that invalid login attempts are counted. When the time period has elapsed, the failed login attempts counter is reset to 0.

Manual Compliance Check: Open PowerShell as an Administrator and execute the following command: *net accounts | findstr -i 'observation'.* Figure 18 shows the output when the command was executed.

```
PS C:\WINDOWS\system32> net accounts | findstr -i 'observation'
Lockout observation window (minutes):                 60
```

Figure 18. Current Running Lockout Observation Window

Figure 18 shows that the Windows host would reset the failed login counter after 60 minutes. This value needs to match the Lockout Observation Window value set in the security baseline to meet compliance.

Automation: The automation of extracting the Lockout Observation Window was achieved with the Ansible task shown in Figure 19.

```
- name: "Retrieve Current Lockout Observation Window"
  win_shell: net accounts | findstr -i 'observation' | %{$_.split(':')[1]} | %{$_ -replace ' ', ''}
  register: current_observation
  changed_when: False
```

Figure 19. Lockout Observation Window Value Extracted

This task extracted the Lockout Observation Window value and stored it into the variable *current_observation* so it could be compared against the security baseline. Figure 20 shows the Ansible task that was used to perform this comparison.

```
- name: "Fail if Lockout Observation Window is not {{ Lockout_ob_window.value }}"
  fail:
    msg: >
     "Security Baseline is not correct. The lockout observation window value found: \
      {{ current_observation.stdout |trim }} is not compliant to the expected: {{ Lockout_ob_window.value }}"
  when: current_observation.stdout |trim |int != Lockout_ob_window.value
```

Figure 20. Comparison Against Security Baseline

The task shown in Figure 20 failed if the comparison did not meet the compliance of the security baseline.

4.4.2 Enforce Password Complexity

Parameter: Password Minimum Length: 14

Description: This parameter defines the minimum length in characters a user account password can be set to.

Manual Compliance Check: Open PowerShell as an Administrator and execute the following command: *net accounts | findstr -i "length" | findstr -i 'Minimum'.* Figure 21 shows the output of the command.

```
PS C:\WINDOWS\system32> net accounts | findstr -i "length" | findstr -i 'Minimum'
Minimum password length:                              0
```

Figure 21. Current Password Minimum Length

The above figure shows the current running configuration in the Windows host for Password Minimum length. This value must meet the value of the security baseline to be deemed compliant.

Automation: Extracting the running configuration for Minimum Password Length was automated with the Ansible task shown in Figure 22.

```
- name: "Retrieve Current Password Length"
  win_shell: net accounts | findstr -i "length" | findstr -i 'Minimum' | %{$_.split(':')[1]} | %{$_ -replace ' ', ''}
  register: current_length
  changed_when: False
```

Figure 22. Ansible Extracting Minimum Password Length

The running configuration was stored in the variable *current_length*. This variable needs to be compared to the security baseline value of 14 and Figure 23 shows how this was done.

```
- name: "Fail if Password Minimum Length is not {{ min_len.value }}"
  fail:
    msg: >
      "Security Baseline is not correct. The password minimum length value \
       found: {{ current_length.stdout |trim }} is not compliant to the expected: {{ min_len.value }}"
  when: current_length.stdout |trim |int < min_len.value
```

Figure 23. Ansible Task Used to Compare Running Configuration to Security Baseline

The task failed if the running configuration did not meet the security baseline.

Parameter: Password Complexity: 1

Description: By setting the password complexity value to 1 it enabled the complexity on the Windows host. When password complexity is enabled on any Windows host, it forces users to have at least three of the following included in the password being set, 1 capital letter, 1 lowercase letter, 1 number, and 1 special character (!?&+). These minimum complexity values meet the recommended security baseline from the NIST recommendations.

Manual Compliance Check: Open PowerShell and execute the command: *secedit /export /cfg <path_to_extact_info>\secpol.cfg.* This command exports some system settings to the file *secpol.cfg* in the designated location as shown in the sample screenshot Figure 24. After the secpol.cfg file was opened, it displayed the Password Complexity parameter indicated with the red outline also displayed in Figure 24.

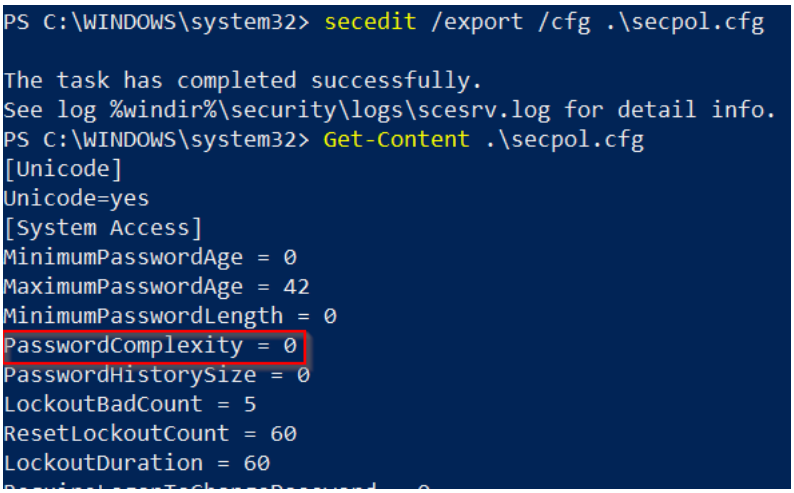

Figure 24. System Settings Exported to File and Password Complexity Displayed

The value for Password Complexity needed to be set to 1 to meet compliance of the security baseline.

Automation: The automation of this required 2 steps. First the system access settings needed to be exported to a file and then the Password Complexity value needed to be extracted from that file. Figure 25 shows how the system access settings were automatically exported.

```
- name: "Export System Access Settings"
  win_shell: secedit /export /cfg C:\Users\project_user\secpol.cfg
```

Figure 25. System Access Settings Exported

Next, the Password Complexity value was extracted from this file and stored to the variable *current_pass_comp_status* which is shown in Figure 26.

```
- name: "Check if Password Complexity is enabled"
  win_shell: Get-Content C:\Users\project_user\secpol.cfg | findstr -i 'PasswordComplexity' | %{$_.split(' ')[2]}
  register: current_pass_comp_status
  changed_when: False
```

Figure 26. Password Complexity Value Extracted

The final step used to check compliance was to compare the current Password Complexity parameter value to the value defined to ensure the parameters of the security baseline were met. This is shown in Figure 27.

```
- name: "Fail if Password Complexity value is not {{ pass_comp.value }} which Indicates it is Disabled"
  fail:
    msg: >
     "Security Baseline is not correct. The password complexity not enabled value \
      found: {{ current_pass_comp_status.stdout |trim }} is not compliant to the expected: {{ pass_comp.value }}"
  when: current_pass_comp_status.stdout |trim |int != pass_comp.value
```

Figure 27. Ansible Task Used to Check the Compliance of Password Complexity

4.4.3 Prevent Excessive Password Ageing

Parameter: Password Maximum Age: 90

Description: This is the maximum number of days that a password is valid. If the password is not changed before the max days expires, the user account is locked.

Manual Compliance Check: Open PowerShell and execute the command: *net accounts | findstr -i 'maximum'.* The output of this command is shown in Figure 28.

```
PS C:\WINDOWS\system32> net accounts | findstr -i 'maximum'
Maximum password age (days):                          42
```

Figure 28. Find the Current Running Configuration for Maximum Password Age

The value must match the parameter defined in the security baseline to be compliant.

Automation: To extract the value from the Windows host, the Ansible task displayed in Figure 29 was used.

```
- name: "Retrieve Current Password Max Age"
  win_shell: net accounts | findstr -i 'maximum' | %{$_.split(':')[1]} | %{$_ -replace ' ', ''}
  register: current_max_age
  changed_when: False
```

Figure 29. Maximum Password Age Extracted

This value was used to compare against the security baseline to determine compliance. The Ansible task used to do this is shown in Figure 30.

```
- name: "Fail if Max Password Age is not {{ max_age.value }}"
  fail:
    msg: >
     "Security Baseline is not correct. The max password age value found: \
      {{ current_max_age.stdout |trim }} is not compliant to the expected: {{ max_age.value }}"
  when: current_max_age.stdout |trim |int != max_age.value
```

Figure 30. Compliance Check for Maximum Password Age

Parameter: Password Minimum Age: 7

Description: This parameter defines the number of days that a password must go without being changed. If a user attempts to change their password during this time period, they will be denied.

Manual Compliance Check: Open PowerShell and execute the command: *net accounts | findstr -i 'maximum' | findstr -i 'age'.* The output of this command is shown in Figure 31.

```
PS C:\WINDOWS\system32> net accounts | findstr -i 'Minimum' | findstr -i 'age'
Minimum password age (days):                          0
```

Figure 31. Checking the Minimum Password Age Value

Automation: The value of Minimum Password Age was extracted with the Ansible task shown in Figure 32.

```
- name: "Retrieve Current Password Min Age"
  win_shell: net accounts | findstr -i 'Minimum' | findstr -i 'age' | %{$_.split(':')[1]} | %{$_ -replace ' ', ''}
  register: current_min_age
  changed_when: False
```

Figure 32. Minimum Password Age Value Extracted

The value was then compared to the security baseline to determine compliance and this is shown in Figure 33.

```
- name: "Fail if Min Password Age is not {{ min_age.value }}"
  fail:
    msg: >
     "Security Baseline is not correct. The min password age value found: \
      {{ current_min_age.stdout |trim }} is not compliant to the expected: {{ min_age.value }}"
  when: current_min_age.stdout |trim |int != min_age.value
```

Figure 33. Minimum Password Age Checked for Compliance

If the value was not compliant, then the task would fail with a message explaining why the value was not compliant.

Parameter: Password History: 10

Description: This parameter defines the number of password changes needed before a password can be reused. It stores a defined number of previously used passwords and when that number is reach and the next password change arrives, the first password is dropped from the list and can be reused.

Manual Compliance Check: Open PowerShell as an Administrator and execute this command: *net accounts | findstr -i 'History'.* Figure 34 shows the output from this command.

```
PS C:\WINDOWS\system32> net accounts | findstr -i 'History'
Length of password history maintained:                    None
```

Figure 34. Finding Password History Value

Automation: The Password History value needed to be extracted so that it could be compared to the security baseline. Figure 35 shows the Ansible task used to extract the value.

```
- name: "Retrieve Current Password History"
  win_shell: net accounts | findstr -i 'History' | %{$_.split(':')[1]} | %{$_ -replace ' ', ''}
  register: current_pass_his
  changed_when: False
```

Figure 35. Password History Value Extracted

This value was used to compare against the security baseline to determine compliance. Figure 36 shows the Ansible task used to do this.

```
- name: "Fail if Password History is not {{ pass_history.value }}"
  fail:
    msg: >
      "Security Baseline is not correct. The password history value found: \
       {{ current_pass_his.stdout |trim }} is not compliant to the expected: {{ pass_history.value }}"
  when: current_pass_his.stdout |trim |int != pass_history.value
```

Figure 36. Password History Compliance Check

The values were compared and if the security baseline was met, the value was compliant. If the value did not meet the security baseline, it was non-compliant.

4.5 Configuration of Security Baseline

The configuration is the second part of the security automation. The security controls are configured to the values that are in line with the security baseline, if any of them are non-compliant. This is handled by the configuration portion of the automation. In the next sections, the manual configuration and the automation of the manual configuration are explained.

4.5.1 Limit Failed Login Attempts

Parameter: Lockout Duration: 60

Manual Configuration: Open PowerShell as an Administrator and run the following command: *net accounts /lockoutduration:60.* The output of the successful configuration change and the new value are shown in Figure 37.

```
PS C:\Users\project_user> net accounts /lockoutduration:60
The command completed successfully.

PS C:\Users\project_user> net accounts
Force user logoff how long after time expires?:       Never
Minimum password age (days):                          7
Maximum password age (days):                          90
Minimum password length:                              14
Length of password history maintained:                10
Lockout threshold:                                    10
Lockout duration (minutes):                           60
Lockout observation window (minutes):                 60
Computer role:                                        WORKSTATION
The command completed successfully.
```

Figure 37. Lockout Duration Configured Manually

Automation: The Ansible task shown in Figure 38 configured the non-compliant parameter to meet the security baseline.

```
- name: "Set Lockout Duration"
  win_shell: "net accounts /lockoutduration:{{ Lockout_duration.value }}"
  when: current_lockout_time.stdout |trim |int != Lockout_duration.value
```

Figure 38. Automation to Configure Lockout Duration

This configuration happened if the when condition in the task evaluated to true meaning that the current running configuration was non-compliant.

Parameter: Lockout Threshold: 10

Manual Configuration: Open PowerShell as an Administrator and run the following command: *net accounts /lockoutthreshold:10.* The output of the successful configuration change and the new value are shown in Figure 39.

```
PS C:\Users\project_user> net accounts /lockoutthreshold:10
The command completed successfully.

PS C:\Users\project_user> net accounts
Force user logoff how long after time expires?:       Never
Minimum password age (days):                          7
Maximum password age (days):                          90
Minimum password length:                              14
Length of password history maintained:                10
Lockout threshold:                                    10
Lockout duration (minutes):                           60
Lockout observation window (minutes):                 60
Computer role:                                        WORKSTATION
The command completed successfully.
```

Figure 39. Lockout Threshold Configured Manually

Automation: The Ansible task shown in Figure 40 configured the non-compliant parameter to meet the security baseline.

```
- name: "Set Lockout Threshold"
  win_shell: "net accounts /lockoutthreshold:{{ Lockout_threshold.value }}"
  when: current_threshold.stdout |trim |int != Lockout_threshold.value
```

Figure 40. Automation to Configure Lockout Threshold

The configuration only executed if the Lockout Threshold parameter is non-compliant to the security baseline.

Parameter: Lockout Observation Window: 60

Manual Configuration: Open PowerShell as an Administrator and run the following command: *net accounts /lockoutwindow:60.* The output of the successful configuration change and the new value are shown in Figure 41.

```
PS C:\Users\project_user> net accounts /lockoutwindow:60
The command completed successfully.

PS C:\Users\project_user> net accounts
Force user logoff how long after time expires?:       Never
Minimum password age (days):                          7
Maximum password age (days):                          90
Minimum password length:                              14
Length of password history maintained:                10
Lockout threshold:                                    10
Lockout duration (minutes):                           60
Lockout observation window (minutes):                 60
Computer role:                                        WORKSTATION
The command completed successfully.
```

Figure 41. Lockout Observation Window Configured Manually

Automation: The Ansible task shown in Figure 42 configured the non-compliant parameter to meet the security baseline.

```
- name: "Set Lockout Observation Window"
  win_shell: "net accounts /lockoutwindow:{{ Lockout_ob_window.value }}"
  when: current_observation.stdout |trim |int != Lockout_ob_window.value
```

Figure 42. Automation of Lockout Observation Window

This task executed when the Lockout Window value was not compliant to the security baseline.

4.5.2 Enforce Password Complexity

Parameter: Minimum Password Length: 14

Manual Configuration: Open PowerShell as an Administrator and run the following command: *net accounts /minpwlen:14.* The output of the successful configuration change and the new value are shown in Figure 43.

```
PS C:\Users\project_user> net accounts /minpwlen:14
The command completed successfully.

PS C:\Users\project_user> net accounts
Force user logoff how long after time expires?:       Never
Minimum password age (days):                          7
Maximum password age (days):                          90
Minimum password length:                              14
Length of password history maintained:                10
Lockout threshold:                                    10
Lockout duration (minutes):                           60
Lockout observation window (minutes):                 60
Computer role:                                        WORKSTATION
The command completed successfully.
```

Figure 43. Manual Configuration of Minimum Password Length

Automation: The Ansible task shown in Figure 44 configured the non-compliant parameter to meet the security baseline.

```
- name: "Set Password Minimum Length"
  win_shell: "net accounts /minpwlen:{{ min_len.value }}"
  when: current_length.stdout |trim |int < min_len.value
```

Figure 44. Automation of Minimum Password Length Configuration

The configuration executed when the security baseline was non-compliant. When the baseline was compliant this tasked was skipped.

Parameter: Password Complexity: 1

Manual Configuration: Open CMD as an Administrator and run the following command: *secedit /export /cfg <Path_to_file>\secpol.cfg.* This command exported the system access settings to the secpol.cfg on the designated file path. The following command: *powershell -command "(GC <Path_to_file>\secpol.cfg) -Replace \"PasswordComplexity = 0\",\"PasswordComplexity = 1\" | Out-File <Path_to_file>\secpol.cfg"* , replaces the non-compliant value of 0 with the compliant value of 1 in the secpol.cfg file. Finally, the command: *secedit /configure*

/db c:\windows\security\local.sdb /cfg C:\Users\project_user\secpol.cfg /areas SECURITYPOLICY , uploads the new value(s) database.

Automation: The Ansible tasks shown in the following Figures 45 and 46 show the tasks that automated this entire process that is mentioned in the manual configuration section. Figure 45 shows how the secpol.cfg was created.

```
- name: "Export System Access Settings"
  win_shell: secedit /export /cfg C:\Users\project_user\secpol.cfg
```

Figure 45. Secpol.cfg File Created with System Access Information

Next, Figure 46 shows how the non-compliant Password Complexity value was changed to the compliant value and how the new settings were uploaded to the database so they would take effect.

```
- block:
    - name: "Enable Password Complexity"
      win_shell: >
        powershell -command "(GC C:\Users\project_user\secpol.cfg) \
        -Replace \"PasswordComplexity = 0\",\"PasswordComplexity = 1\" | Out-File C:\Users\project_user\secpol.cfg"
      args:
        executable: cmd

    - name: "Load new setting for Password Complexity"
      win_shell: >
       secedit /configure /db c:\windows\security\local.sdb /cfg C:\Users\project_user\secpol.cfg /areas SECURITYPOLICY
      args:
        executable: cmd
  when: current_pass_comp_status.stdout |trim |int != pass_comp.value
```

Figure 46. Automation of Password Complexity Configuration

The configuration of these tasks was stored in a separate block and only executed when the evaluating condition was true.

4.5.3 Prevent Excessive Password Ageing

Parameter: Password Maximum Age: 90

Manual Configuration: Open PowerShell as an Administrator and run the following command: *net accounts /maxpwage:90.* The output of the successful configuration change and the new value are shown in Figure 47.

```
PS C:\Users\project_user> net accounts /maxpwage:90
The command completed successfully.

PS C:\Users\project_user> net accounts
Force user logoff how long after time expires?:       Never
Minimum password age (days):                          7
Maximum password age (days):                          90
Minimum password length:                              14
Length of password history maintained:                10
Lockout threshold:                                    10
Lockout duration (minutes):                           60
Lockout observation window (minutes):                 60
Computer role:                                        WORKSTATION
The command completed successfully.
```

Figure 47. Manual Configuration of Maximum Password Age

Automation: The Ansible task shown in Figure 48 configured the non-compliant parameter to meet the security baseline.

```
- name: "Set Max Password Age"
  win_shell: "net accounts /maxpwage:{{ max_age.value }}"
  when: current_max_age.stdout |trim |int != max_age.value
```

Figure 48. Automation of Maximum Password Age Configuration

This configuration occurred when the security baseline was non-compliant.

Parameter: Password Minimum Age: 7

Manual Configuration: Open PowerShell as an Administrator and run the following command: *net accounts /minpwage:7.* The output of the successful configuration change and the new value are shown in Figure 49.

```
PS C:\Users\project_user> net accounts /minpwage:7
The command completed successfully.

PS C:\Users\project_user> net accounts
Force user logoff how long after time expires?:       Never
Minimum password age (days):                          7
Maximum password age (days):                          90
Minimum password length:                              14
Length of password history maintained:                10
Lockout threshold:                                    10
Lockout duration (minutes):                           60
Lockout observation window (minutes):                 60
Computer role:                                        WORKSTATION
The command completed successfully.
```

Figure 49. Manual Configuration of Minimum Password Age

Automation: The Ansible task shown in Figure 50 configured the non-compliant parameter to meet the security baseline.

```
- name: "Set Min Password Age"
  win_shell: "net accounts /minpwage:{{ min_age.value }}"
  when: current_min_age.stdout |trim |int != min_age.value
```

Figure 50. Automation of Minimum Password Configuration

This configuration occurred when the security baseline was non-compliant.

Parameter: Password History: 10

Manual Configuration: Open PowerShell as an Administrator and run the following command: *net accounts /uniquepw:10.* The output of the successful configuration change and the new value are shown in Figure 51.

```
PS C:\Users\project_user> net accounts /uniquepw:10
The command completed successfully.

PS C:\Users\project_user> net accounts
Force user logoff how long after time expires?:       Never
Minimum password age (days):                          7
Maximum password age (days):                          90
Minimum password length:                              14
Length of password history maintained:                10
Lockout threshold:                                    10
Lockout duration (minutes):                           60
Lockout observation window (minutes):                 60
Computer role:                                        WORKSTATION
The command completed successfully.
```

Figure 51. Manual Configuration of Password History

Automation: The Ansible task shown in Figure 52 configured the non-compliant parameter to meet the security baseline.

```
- name: "Set Password History"
  win_shell: "net accounts /uniquepw:{{ pass_history.value }}"
  when: current_pass_his.stdout |trim |int != pass_history.value
```

Figure 52. Automation of Password History Configuration

This configuration occurred when the security baseline was non-compliant.

4.6 Verify Hardening of Password Policy for Windows Server

The following sections show the results of the implementations of the security baseline. The controls are meaningless if they do not take effect to help protect the system. The verifications below provide proof that the security controls are protecting the system.

4.6.1 Verify Enforcement of Limit Failed Login Attempts Control

Parameters: Lockout Duration and Lockout Observation Window

Verification: For the purposes of verifying these portions of the control, the lockout duration time was set to a value of 1 minute. The lockout threshold was set to three (3) attempts and the lockout observation window was set to 1 minute. Figure 53 shows the logs verifying that these portions of the control took effect.

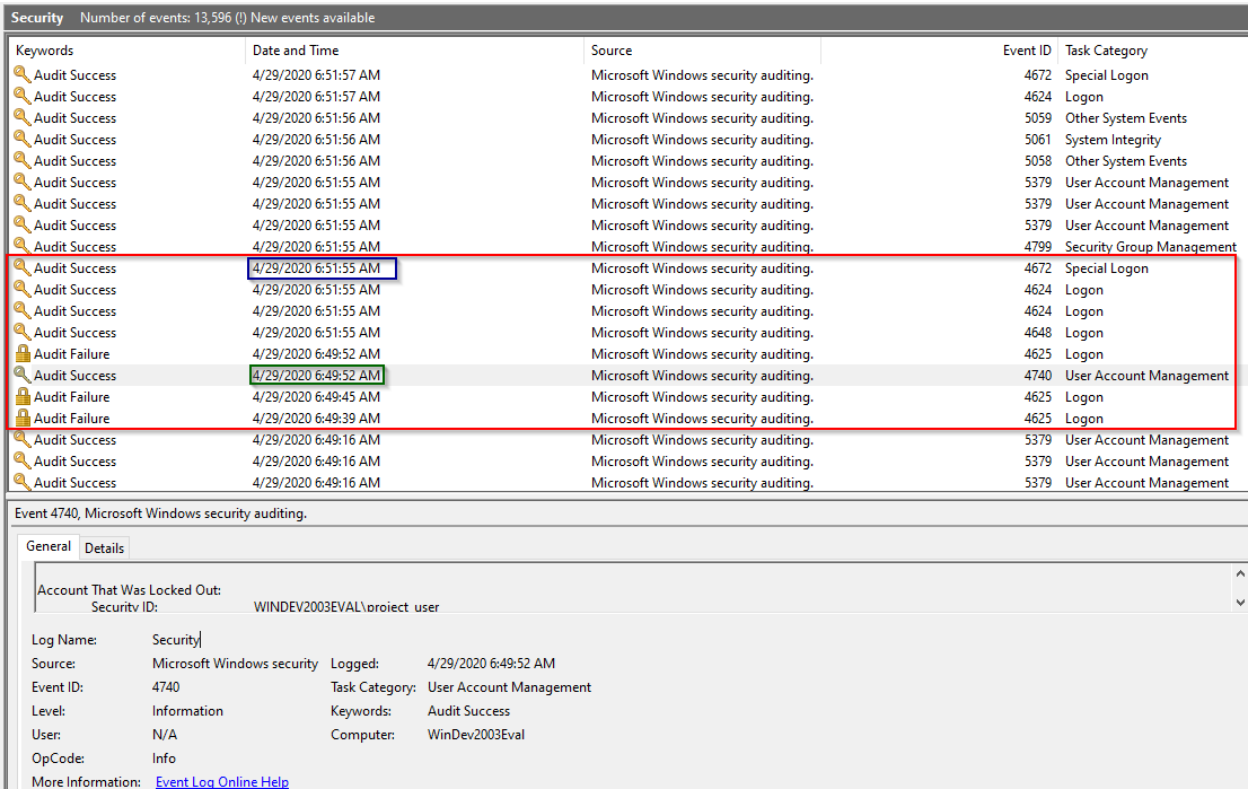

Figure 53. Lockout Duration Taking Effect

The above figure indicates the series of events that occurred on the system. The red box shows the failed login attempts of 3, the account lockout and the account

restoration/successful login that occurred. More specifically, the green box indicates the project_user's account being locked after 3 failed attempts. The blue box shows the event of a successful logon by an Administrator which the project_user is, approximately 1.5 minutes later. It's safe to say that this control is working because the lockout observation window has to be set to a value that is equal to the lockout duration. Both values were set to 1 minute indicating that the lockout duration and account unlock occurred simultaneously.

Parameter: Lockout Threshold

Verification: The test for the Lockout Threshold was tried against the project_user that was created. If the control is taking effect, the project_user account should only allow 10 password attempts. On the 11th failed password attempt, the project_user's account should be locked for the Lockout Duration time and then the account should be automatically unlocked. Figure 54 shows that the project_user's account was indeed locked after 10 failed password attempts.

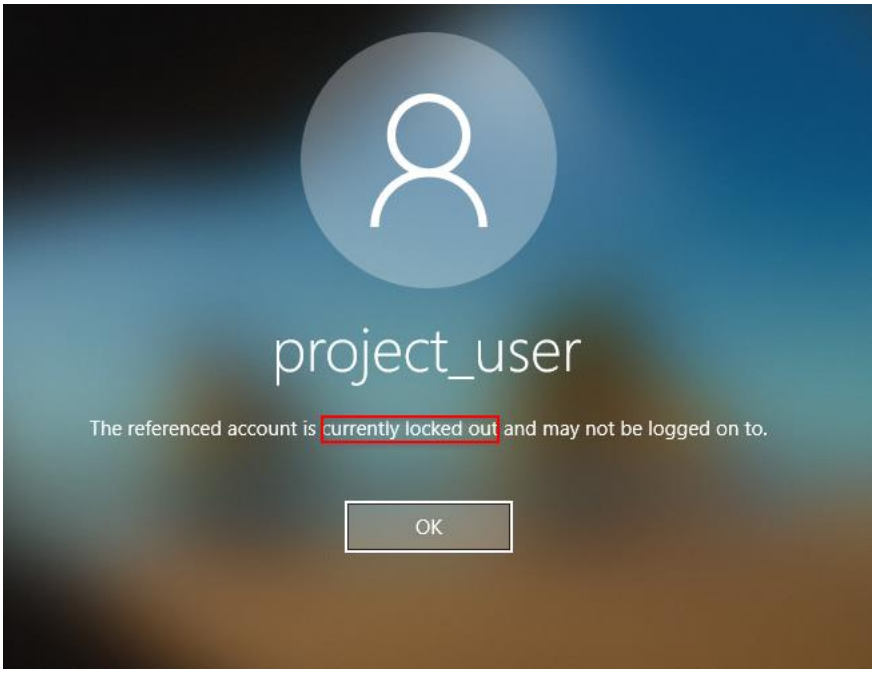

Figure 54. User Account Locked After Lockout Threshold Violated

The Windows Event Viewer provided more detailed information about the effect of the security control. Figure 54 shows the system events that were logged.

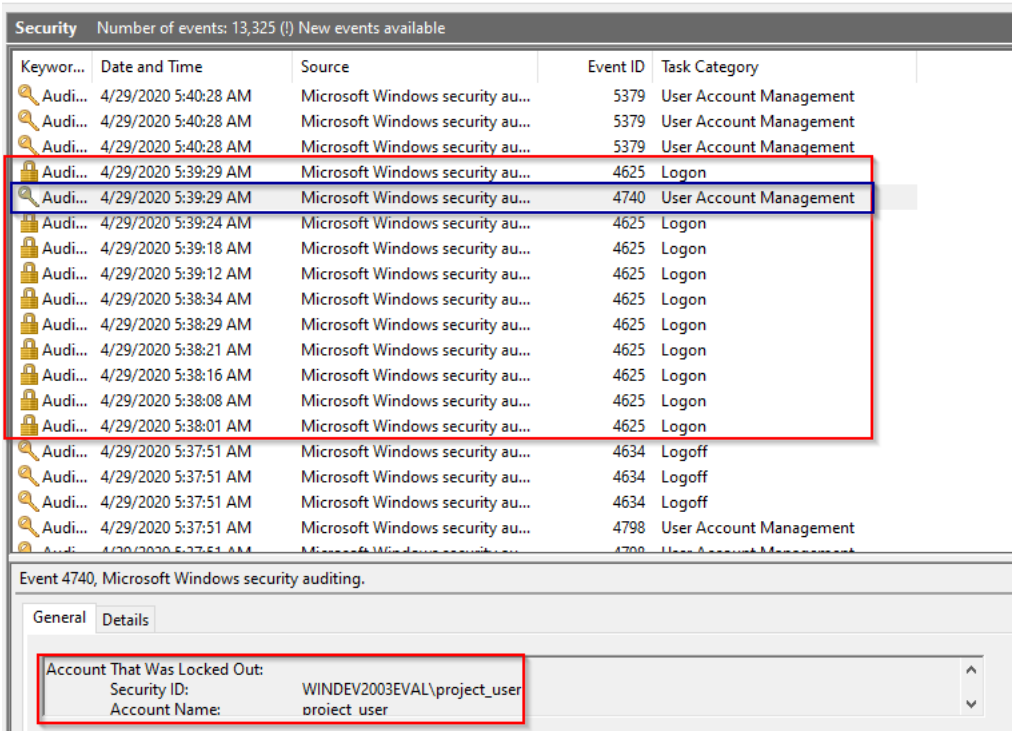

Figure 55. Lockout Threshold Taking Effect

The Event ID: 4625 in the logs indicate a failed logon attempt. If the attempt had been successful, it would have been logged with Event ID 4624. The image shows a series of 10 logon attempts and then there is the Event ID 4740 which indicates an account being locked. On the lower part of the Figure, there is more information about Event 4740 explaining that the user project_user was locked out. This proves that this portion of the Limit Failed Login Attempts control is active and working as expected.

4.6.2 Verify Enforcement of Password Complexity and Length Control

Parameters: Password Minimum Length and Password Complexity

Verification: For this verification, an attempt to set a password that was shorter than 14 characters but had 1 lowercase, 1 uppercase, 1 number, and 1 special character was made and an attempt of a password that was longer than 14 characters but only contained 1 uppercase and 1 lowercase. Figure 56 shows the result of this attempt.

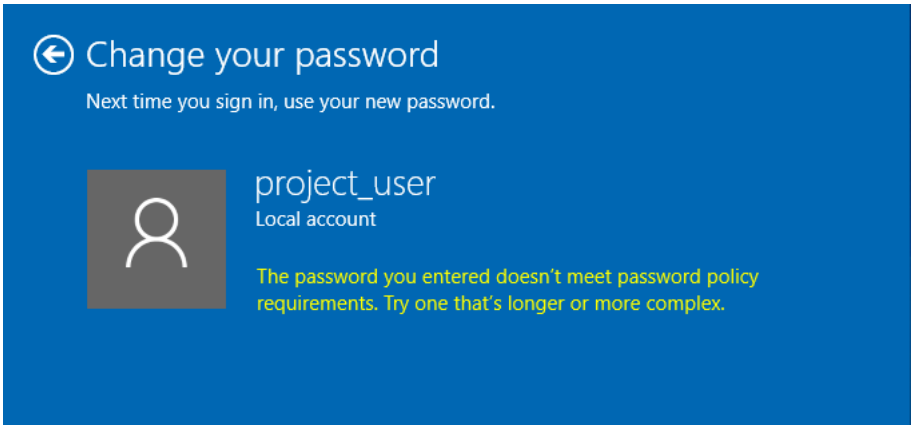

Figure 56. Password Min Length and Password Complexity Verification

In both cases, the system would not accept the password because it was not long enough or not long enough and complex enough.

4.6.3 Verify Enforcement Preventing Excessive Password Ageing

Parameter: Password Maximum Age

Verification: This control was not verified because the maximum lifetime was set to 90 days. It is somewhat safe to say that this control took effect because all other controls in the password policy took effect. If the user would not change their password before the max lifetime expired, their account would be locked, and an Administrator would need to unlock their account. They would also be forced to change their password at next logon.

Parameter: Password Minimum Age

Verification: Verifying this portion of the control was a matter of checking to see if the password could be changed before the 7-day minimum password lifetime had passed. To be sure there were no other password controls taking effect, the password complexity was disabled, and the password minimum length was 14. The test password used for the attempted change was *!Big662m@ch$negunFunk.* This password satisfied the minimum length required so the only parameter that could block a password change was the Password Minimum Lifetime 7. Figure 56 shows the message when trying to change the password.

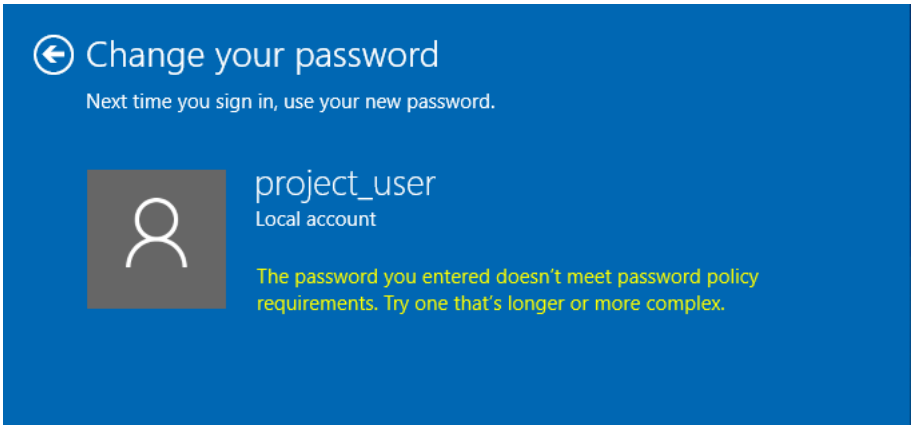

Figure 57. Password Minimum Lifetime Verified

Because the Password Minimum Lifetime was set to 7, this means that 7 days needed to pass before the password could be changed.

Parameter: Password History

Verification: To verify this portion of the control, the password min days was changed to 0. The password history was set to 10. This means that the current password would not be usable until the 11th change. After 2 password changes, an attempt to use the original password was tried and Figure 58 shows the outcome.

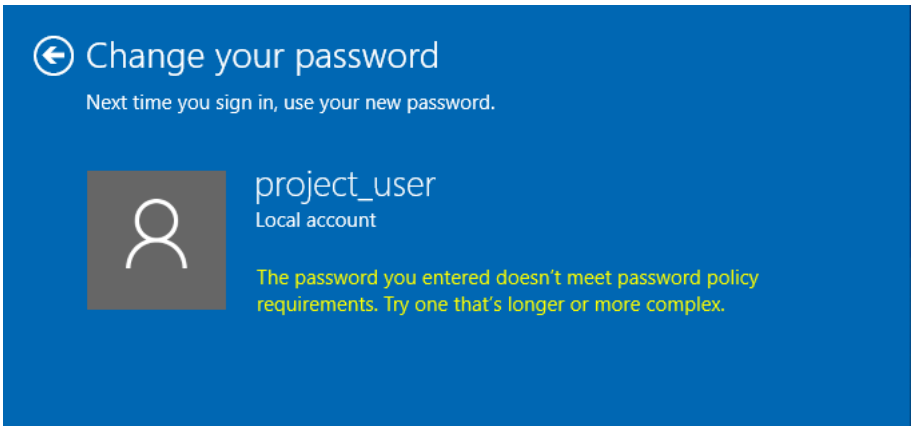

Figure 58. Password History Verification

After changing the password 10 times, my original password was able to be used again. This shows that the security control was working as expected.

5 Discussion

5.1 Project Conclusions

This project came to a successful ending. The security baseline was established, and the automation of the baseline was built and implemented. Having a compliance check ensured that the baseline could be verified at any time, and there were explicit messages that displayed how the baseline was violated. The configuration allowed for a quick fix of the baseline if needed and kept the system compliant to the baseline. Configuration management is a viable tool for security automation. The idea behind using configuration management is to keep the system at a desired state, which also applies to security. Security professionals want to keep the system in a state where the CIA is protected. It is clear to see that this can also be achieved using configuration management tools.

The project scope was limited to the password policy of the Windows host; however, the overall limitations are minimal. Ansible is a very well maintained and documented configuration management tool so its usefulness will not fade away anytime soon. This means that Ansible could be used to implement a full security baseline on a Windows host going beyond the password policy. For these reasons, Ansible is a viable tool for security automation presently and in the future. The questions this project aimed to answer were: Can configuration management tools be used to automate security on Windows hosts? and Is Ansible a viable configuration management tool for implementing security automation on Windows hosts?. The information in this document clearly confirms that CM tools; Ansible specifically, can be used to automate security on Windows hosts.

References

Red Hat. 2019. Ansible Documentation. Accessed on 18 March 2020. Retrieved from https://docs.ansible.com/ansible/latest/index.html

Cropley, A. J. 2019. Qualitative research methods: A practice-oriented introduction for students of psychology and education. Riga, Latvia: Zinātne

Fice, G. & Waller, J. July 2012. Benchmarking: Key tools to develop your understanding and use of benchmarking. Accessed on 18 March 2020. Retrieved from https://www.jisc.ac.uk/full-guide/benchmarking

Heidi, E. May 2019. An Introduction to Configuration Management. Accessed on 18 March 2020. Retrieved from https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management

Henderson, B. 2018. Inside The Playbook: Connecting to A Windows Host. Accessed on 18 March 2020. Retrieved from https://www.ansible.com/blog/connecting-to-a-windows-host

Mack, N. & Woodsong, C. 2005. Qualitative Research Methods Overview. Family Health International.; United States. Agency for International Development. North Carolina : FLI USAID

Microsoft. 2020. What is a virtual machine? Accessed on 18 March 2020. Retrieved from https://azure.microsoft.com/en-us/overview/what-is-a-virtual-machine/

NIST. 2020. Digital Identity Guidelines: Authentication and Lifecycle Management. Accessed on 18 March 2020. Retrieved from https://pages.nist.gov/800-63-3/sp800-63b.html#throttle

NIST. 2019. New To The Framework. Accessed on 18 March 2020. Retrieved from https://www.nist.gov/cyberframework/new-framework

NIST. 2015. Security and Privacy Controls for Federal Information Systems and Organizations. Accessed on 18 March 2020. Retrieved from https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf

Red Hat. 2020. What is Virtualization? Accessed on 18 March 2020. Retrieved from https://www.redhat.com/en/topics/virtualization/what-is-virtualization

Showkat, N. & Parveen, H. 2017. In-depth Interview. e-PG Pathshala (UGC & MHRD)

VirtualBox. 2020. Welcome to VirtualBox.org. Accessed on 18 March 2020. Retrieved from https://www.virtualbox.org/

Wheeler, S. 2020. About Execution Polices. Accessed on 18 March 2020. Retrieved from https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_execution_policies?view=powershell-7